U.S. Patent No. 6,362,822

Unreal Engine 4

Activision
Exhibit 3

Claim 1

Claim 1	Unreal Engine 4
[1.1] A shadow rendering method for use in a computer system, the method comprising the steps of:	The Unreal Engine 4 ("Unreal4") is a cross-platform video game engine developed by Epic Games and used in video games published by Activision. Unreal4 utilizes rendering methods such as deferred shading/lighting and cascaded shadow maps for example, as shown below: Choose your OS: **Rendering Overview**  The rendering system in Unreal Engine 4 is an all-new, DirectX 11 pipeline that includes deferred shading, global illumination, lit translucency, and post processing as well as GPU particle simulation utilizing vector fields. **Deferred Shading** All lights are applied deferred in Unreal Engine 4, as opposed to the forward lighting path used in Unreal Engine 3. Materials write out their attributes into the GBuffers, and lighting passes read in the per-pixel material properties and perform lighting with them. https://docs.unrealengine.com/latest/INT/Engine/Rendering/Overview/

Claim 1	Unreal Engine 4

Cascaded Shadow Maps

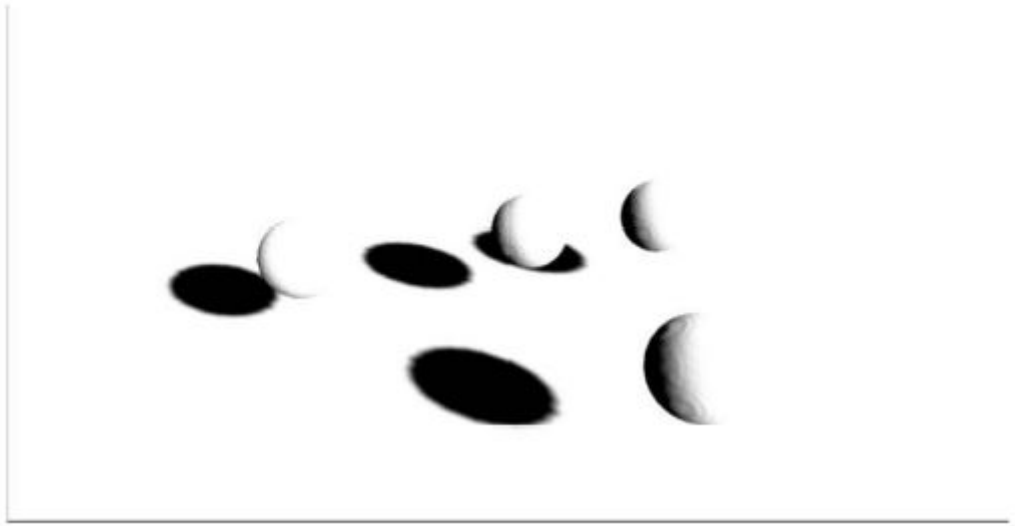
Property	Description
Dynamic Shadow Distance MovableLight	How far **Cascaded Shadow Map** dynamic shadows will cover for a movable light, measured from the camera.
Dynamic Shadow Distance StationaryLight	How far **Cascaded Shadow Map** dynamic shadows will cover for a stationary light, measured from the camera.
Num Dynamic Shadow Cascades	Number of cascades to split the view frustum into for the whole scene.
Cascade Distribution Exponent	Controls whether the cascades are distributed closer to the camera (larger exponent) or further from the camera (smaller exponent).
Cascade Transition Fraction	Proportion of the fade region between cascades.
Shadow Distance Fadeout Fraction	Controls the size of the fade out region at the far extent of the dynamic shadow's influence.
Use Inset Shadows for Movable Objects	(Stationary lights only) Whether to use per-object inset shadows for movable components, even though cascaded shadow maps are enabled.

https://docs.unrealengine.com/en-us/Engine/Rendering/LightingAndShadows/LightTypes/Directional

Claim 1	Unreal Engine 4
[1.2] providing observer data of a simulated multi-dimensional scene;	Activision performs the step of "providing observer data of a simulated multi-dimensional scene." The '822 specification teaches that, in one embodiment, observer data may include "observed color data and observed depth data associated with a plurality of modeled polygons within the scene as rendered from an observer's perspective." Col. 3:38-41. Observed color data includes, for example, "an observed red-green-blue value for the pixel(s)," and observed depth data includes, for example, "an observed z-buffer value for the pixel(s)." Col. 3:43-46. In the context of 3D graphics, a "camera" observes one or more objects in a simulated world. The camera captures a particular viewpoint of the world, observing specific data associated with the objects as seen from the camera's point-of-view. Unreal Engine 4's deferred shading technique uses a geometry buffer (GBuffer) that stores material and object attributes as shown, for example, below: Using GBuffer Properties A GBuffer consists of multiple textures that store material (e.g. subsurface/specular color, roughness, ...) and object attributes (e.g. normal, depth) without lighting to compute shading (how light interacts with a material). In a deferred renderer, we first render the GBuffer and then compute all lighting (deferred) with the GBuffer attributes. If UE4 uses the deferred shading path (e.g. DirectX 11 or high end OpenGL), we can get access to those buffers during post processing. https://docs.unrealengine.com/en-us/Engine/Rendering/PostProcessEffects/PostProcessMaterials

Claim 1	Unreal Engine 4
	Creating the GBuffer Deferred shading uses the concept of a "GBuffer" (Geometry Buffer) which is a series of render targets that store different bits of information about the geometry such as the world normal, base color, roughness, etc. Unreal samples these buffers when lighting is calculated to determine the final shading. Before it gets there though, Unreal goes through a few steps to create and fill it. The exact contents of the GBuffer can differ, the number of channels and their uses can be shuffled around depending on your project settings. A common case example is a 5 texture GBuffer, A through E. `GBufferA.rgb = World Normal`, with `PerObjectGBufferData` filling the alpha channel. `GBufferB.rgba = Metallic, Specular, Roughness, ShadingModelID`. `GBufferC.rgb` is the `BaseColor` with `GBufferAO` filling the alpha channel. `GBufferD` is dedicated to custom data and `GBufferE` is for precomputed shadow factors. https://medium.com/@lordned/unreal-engine-4-rendering-part-4-the-deferred-shading-pipeline-389fc0175789 The geometry buffers ("GBuffer") contain "observer data" described in the spec of the '822 Patent such as "rgb" or red, blue, and green values for the given rendered scene (i.e. "GBufferA.rgb = World Normal" as illustrated above), including further details such as the shape and surface material of objects rendered in the scene. For example, as illustrated above, the GBuffer can contain observed color data such as the rednered objects, color, shape, and material characteristics i.e. how rough or "shiny" an object may appear based on the material it is "made" from. Thus, Unreal4 provides observed color data.

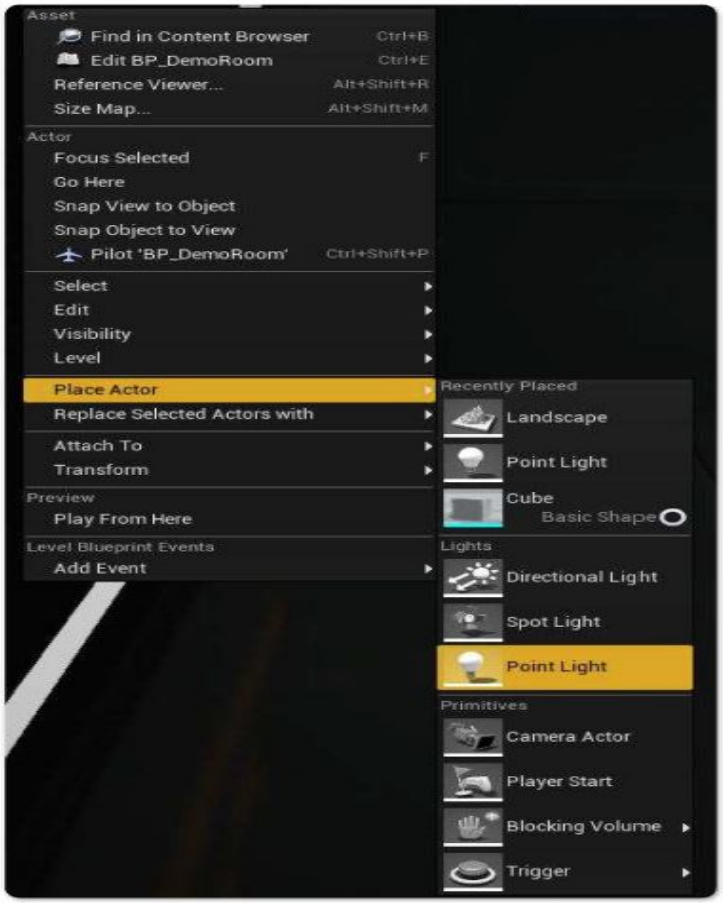
Claim 1	Unreal Engine 4
	Inside *BasePassPixelShader.usf* the `FPixelShaderInOut MainPS` function acts as the entry point for the pixel shader. This function looks quite complicated due to the numerous preprocessor defines but is mostly filled with boilerplate code. Unreal uses several different methods to calculate the required data for the GBuffer depending on what lighting model and features you have enabled. Unless you need to change some of this boilerplate code, the first significant function is partway down where the shader gets the values for `BaseColor`, `Metallic`, `Specular`, `MaterialAO`, and `Roughness`. It does this by calling the functions declared in *MaterialTemplate.ush* and their implementations are defined by your material graph. https://medium.com/@lordned/unreal-engine-4-rendering-part-4-the-deferred-shading-pipeline-389fc0175789 The GBuffer populates the values for the observed data from a MaterialTemplate.ush

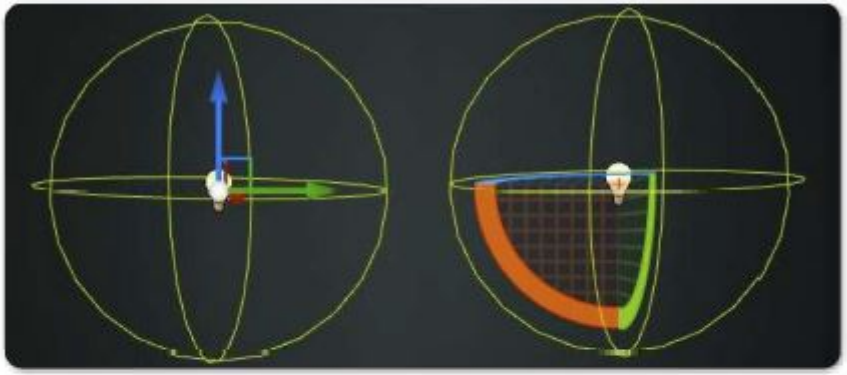
Claim 1	Unreal Engine 4
	Shadowed and Unshadowed Lights Unreal draws lighting in multiple stages. Non shadow-casting lights are drawn first, and then indirect lighting (via light propegation volumes) is drawn. Finally Unreal draws all shadow casting lights. Unreal uses similar pixel shaders for shadow casting and non-shadow casting lights —the difference between them comes from additional pre-processing steps for shadow casting lights. For each light, Unreal computes a *ScreenShadowMaskTexture* which is a screenspace representation of the shadowed pixels in your scene.  A ScreenShadowMaskTexture for a simple scene with some spheres in it https://medium.com/@lordned/unreal-engine-4-rendering-part-4-the-deferred-shading-pipeline-

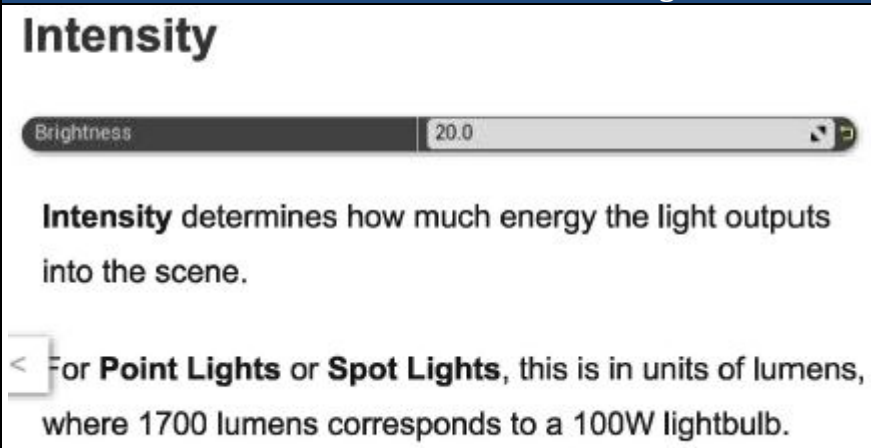
Claim 1	Unreal Engine 4
	389fc0175789 To do this, Unreal renders geometry that appears to be matched to the bounding box of each object in your scene, and geometric representations of objects in your scene. It *does not re-render the objects in your scene* and instead samples the GBuffer combining the depth of a given pixel to see if it would be in the way of a cast light shadow. Sound complicated? Don't worry, it is. The good news is that the only takeaway we need here is that each shadowed light computes a screenspace representation of what surfaces are in shadow and this data is used later! https://medium.com/@lordned/unreal-engine-4-rendering-part-4-the-deferred-shading-pipeline-389fc0175789 As explained above, rather then "re-rendering" the shadows of rendered objects, the shadows rendered in the scene are rendered based off information contained in the GBuffer that is "combin[ed] [with] the ***depth of a given pixel*** to see if it would be in the way of a cast light shadow." (emphasis added). This is an example of Unreal4 performing calculations based on "depth data" of a given pixel.

Claim 1	Unreal Engine 4
[1.3] providing lighting data associated with a plurality of simulated light sources arranged to illuminate said scene, said lighting data including light image data;	Activision performs the step of "providing lighting data associated with a plurality of simulated light sources arranged to illuminate said scene, said lighting data including light image data." "Light image **51C** includes RGB pixel data values for the light emitted, for X by Y number of pixels. For example, the data in light image **51C** can represent the intensity, color, and/or pattern of light emitted by light source #1." Col. 7:15-19. Unreal4 supports a variety of lights, such as directional lights, spot lights, and point lights, for example, as shown below: Rendering Overview  The rendering system in Unreal Engine 4 is an all-new, DirectX 11 pipeline that includes deferred shading, global illumination, lit translucency, and post processing as well as GPU particle simulation utilizing vector fields. Deferred Shading All lights are applied deferred in Unreal Engine 4, as opposed to the forward lighting path used in Unreal Engine 3. Materials write out their attributes into the GBuffers, and lighting passes read in the per-pixel material properties and perform lighting with them. Lighting Paths There are three lighting paths in UE4: • **Fully dynamic** - with Movable Lights • **Partially static** - with Stationary Lights • **Fully static** - with Static Lights

Claim 1	Unreal Engine 4
	https://docs.unrealengine.com/en-us/Engine/Rendering/Overview Lighting Basics Choose your OS:  On this page: • Placing Lights • Intensity • Light Color • Attenuation Radius • Source Radius and Length In Unreal Engine 4, there are a number of ways to add lights to a scene and there are a few key properties that have the greatest impact on lighting in the world. https://docs.unrealengine.com/en-US/Engine/Rendering/LightingAndShadows/Basics

Claim 1	Unreal Engine 4
	Lighting Basics | Unreal Engine  Once a light is added, you can then adjust the position and rotation of the light using the position (**W**) and rotation (**E**) widgets like any other object. https://docs.unrealengine.com/en-US/Engine/Rendering/LightingAndShadows/Basics

Claim 1	Unreal Engine 4
	Once a light is added, you can then adjust the position and rotation of the light using the position (**W**) and rotation (**E**) widgets like any other object.  Lights are represented by these sprites in the editor. From left to right they are: **Point Light**, **Spot Light**, and **Directional Light**.  https://docs.unrealengine.com/en-US/Engine/Rendering/LightingAndShadows/Basics

Claim 1	Unreal Engine 4
	 https://docs.unrealengine.com/en-US/Engine/Rendering/LightingAndShadows/Basics A light within a scene is associated with lighting data such as, for example, color (*see, e.g.*, "RGB diffuse color") and intensity. *See, e.g.*, below.

Claim 1	Unreal Engine 4
	Light Color  **Light Color** will adjust the color of the light and the sprite that represents the light in the editor will change its color to match  https://docs.unrealengine.com/en-US/Engine/Rendering/LightingAndShadows/Basics As explained in Claim 1.4 below, for example, a light is also associated with depth data, such as for the purpose of determining which objects are illuminated by said light.

Claim 1	Unreal Engine 4
[1.4] for each of said plurality of light sources, comparing at least a portion of said observer data with at least a portion of said lighting data to determine if a modeled point within said scene is illuminated by said light source and storing at least a portion of said light image data associated with said point and said light source in a light accumulation buffer; and then	"[F]or each of said plurality of light sources," Activision performs the step of "comparing at least a portion of said observer data with at least a portion of said lighting data to determine if a modeled point within said scene is illuminated by said light source and storing at least a portion of said light image data associated with said point and said light source in a light accumulation buffer." Upon information and belief, Unreal4 iterates through each light source to determine whether a modeled point is illuminated by, for example, intersecting light sources with portions of the screen to be rendered. This intersection process involves comparing the observed depth data with the lighting depth data. As shown below, for example, the Unreal4 then "[a]ccumulate[s] this lighting and stores it into a Buffers" or "Light accumulation texture[s]." **Base Pass Pixel Shader** Now that we know shadowed lights create a screenspace shadow texture we can go back to looking at how the base pass pixel shader works. As a reminder, this is run for each light in the scene so for any object that has multiple lights affecting it it will be run multiple times per pixel. The pixel shader can be quite simple, we'll be interested more in the functions this pixel shader calls. https://medium.com/@lordned/unreal-engine-4-rendering-part-4-the-deferred-shading-pipeline-389fc0175789

Claim 1	Unreal Engine 4

```
void RadialPixelMain( float4 InScreenPosition, float4 SVPos,
out float4 OutColor)
{
// Intermediate variables have been removed for brevity
FScreenSpaceData ScreenSpaceData =
GetScreenSpaceData(ScreenUV);
FDeferredLightData LightData =
SetupLightDataForStandardDeferred();

OutColor = GetDynamicLighting(WorldPosition, CameraVector,
ScreenSpaceData.GBuffer, ScreenSpaceData.AmbientOcclusion,
ScreenSpaceData.GBuffer.ShadingModelID, LightData,
GetPerPixelLightAttenuation(ScreenUV), Dither, Random);

OutColor *= ComputeLightProfileMultiplier(WorldPosition,
DeferredLightUniforms_LightPosition,
DeferredLightUniforms_NormalizedLightDirection);
}
```

There's only a couple of functions so we'll hop through what each one does. `GetScreenSpaceData` retrieves the information from the GBuffer for a given pixel. `SetupLightDataForStandardDeferred` calculates information such as the light direction, light color, falloff, etc. Finally, it calls `GetDynamicLighting` and passes in all of the data we've calculated so far—where the pixel is, what the GBuffer data is, what Shading Model ID to use, and our light's information.

https://medium.com/@lordned/unreal-engine-4-rendering-part-4-the-deferred-shading-pipeline-389fc0175789

Claim 1	Unreal Engine 4
	Now that we've determined the shadow factor for both surface and subsurface data we calculate light attenuation. Attenuation is effectively the falloff in energy based on the distance from the light and can be modified to create different effects, ie: Toon shading often removes Attenuation from the calculation so that your distance to a light source doesn't matter. Unreal calculates `SurfaceAttenuation` and `SubsurfaceAttenuation` separately based on distance, light radius and falloff, and our shadow term. **Shadowing is combined with attenuation**, which means our future calculations only take *attenuation strength* into account. Finally we calculate our Surface Shading for this pixel. **Surface Shading takes the GBuffer, Surface Roughness, Area Light Specular, Light Direction, View Direction, and Normal** into account. Roughness is determined by our GBuffer data. Area Light Specular uses physically based rendering (based on our light data and roughness) to calculate a new energy value and can modify the roughness and light vector. https://medium.com/@lordned/unreal-engine-4-rendering-part-4-the-deferred-shading-pipeline-389fc0175789

Claim 1	Unreal Engine 4
	Accumulated Light Because the *BasePassPixelShaders* are run for every light that affects an object, Unreal accumulates this lighting and stores it in a buffer. This buffer isn't even drawn to the screen until several steps later in the `ResolveSceneColor` step. Several additional things are calculated before that such as translucent objects (which are drawn using traditional forward rendering techniques), screen space temporal anti aliasing and screen space reflections. https://medium.com/@lordned/unreal-engine-4-rendering-part-4-the-deferred-shading-pipeline-389fc0175789

Claim 1	Unreal Engine 4
	Surface Shading finally gives us a chance to modify how each surface responds to this data. This function is located in *ShadingModels.ush* and is just a big switch statement that looks at our ShadingModel ID that was written into the GBuffer way earlier! Many of the lighting models share a standard shading function, but some of the more unusual shading models use their own custom implementations. **Surface Shading does not take attenuation into account**, so it only deals with calculating the color of the surface without shadows. Attenuation (which is distance + shadow) isn't taken into account until the Light Accumulator is run. The Light Accumulator takes the surface lighting and attenuation into account and adds together surface and sub-surface lighting correctly after multiplying them by the light attenuation value. Finally the Dynamic Lighting function returns the total light accumulated by the Light Accumulator. In practice this is just surface + subsurface lighting but the code is complicated by subsurface properties and debug options. https://medium.com/@lordned/unreal-engine-4-rendering-part-4-the-deferred-shading-pipeline-389fc0175789

Claim 1	Unreal Engine 4
[1.5] combining at least a portion of said light accumulation buffer with said observer data; and	Activision performs the step of "combining at least a portion of said light accumulation buffer with said observer data." On information and belief, Unreal4 combines the accumulated light with the observer data such as, for example, in the "`ResolveSceneColor` step" as shown below: **Accumulated Light** Because the *BasePassPixelShaders* are run for every light that affects an object, Unreal accumulates this lighting and stores it in a buffer. This buffer isn't even drawn to the screen until several steps later in the `ResolveSceneColor` step. Several additional things are calculated before that such as translucent objects (which are drawn using traditional forward rendering techniques), screen space temporal anti aliasing and screen space reflections. https://medium.com/@lordned/unreal-engine-4-rendering-part-4-the-deferred-shading-pipeline-389fc0175789
[1.6] displaying resulting image data to a computer screen.	On information and belief, Activision performs the step of "displaying resulting image data to a computer screen," such as, for example, when testing a video game running the Unreal 4 Engine such as Ghostbusters (2016). As shown below, for example, Activision displays resulting image data to a computer screen.

Claim 1	Unreal Engine 4
	 http://ghostbusters.wikia.com/wiki/Ghostbusters_Activision_Video_Game_(2016)